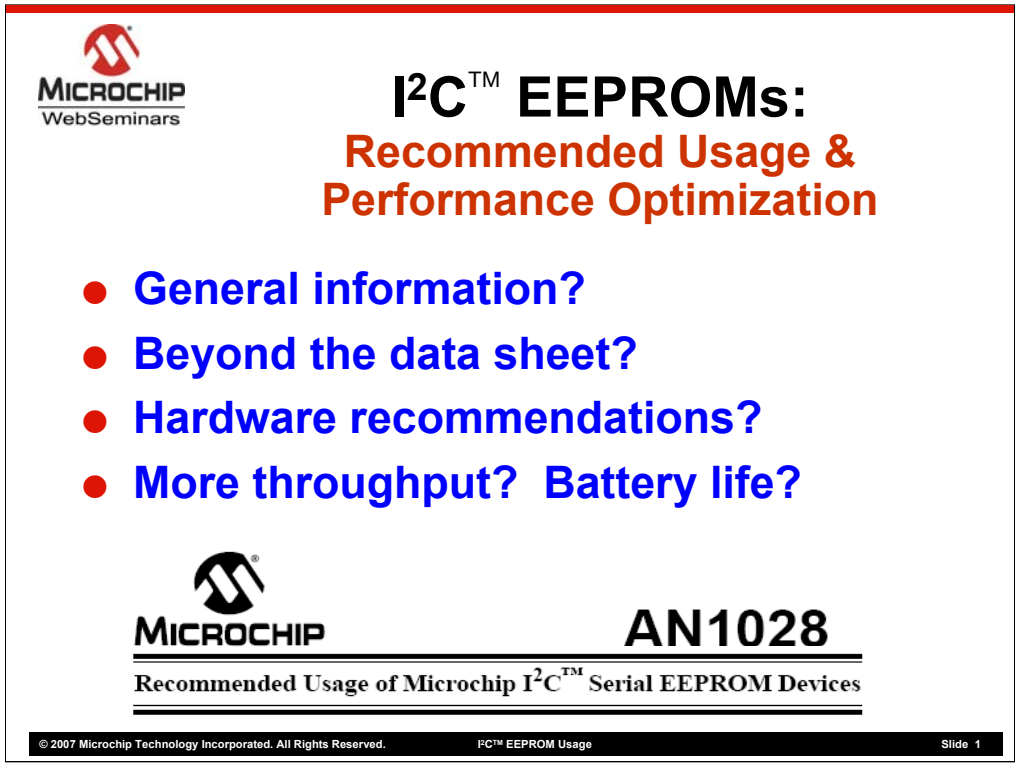


Hi, my name is Barry Blixt, marketing manager for Microchip memory products. Welcome to this 20-minute web seminar in which we will discuss some recommended usage practices for I²C™ serial EEPROMs. We will also discuss some ways to optimize the performance of an I²C system.

So, what specific information will we cover?

•First, are you looking for some general information on the protocol itself? We will discuss the advantages of the I²C bus.

•Are you designing an I²C system and looking for more information than you can find on the data sheet? This web seminar will build upon the information available in a typical data sheet.

•Do you need some details about the best way to design a board? We will discuss several hardware suggestions.

•Do you need to increase data throughput in your system? Or is battery life your major concern? We will talk about ways to fine tune your design.

Much of the information in this seminar is taken from Microchip data sheets as well as our application note AN 1028 entitled “Recommended Usage of Microchip I²C Serial EEPROM devices.”

Now, let’s look at our agenda.

MICROCHIP WebSeminars

Agenda

- **I²C™ Benefits**
- **Hardware Recommendations**
- **Firmware Performance Enhancements**
- **For More Information**

 I²C™ EEPROM Usage Slide 2

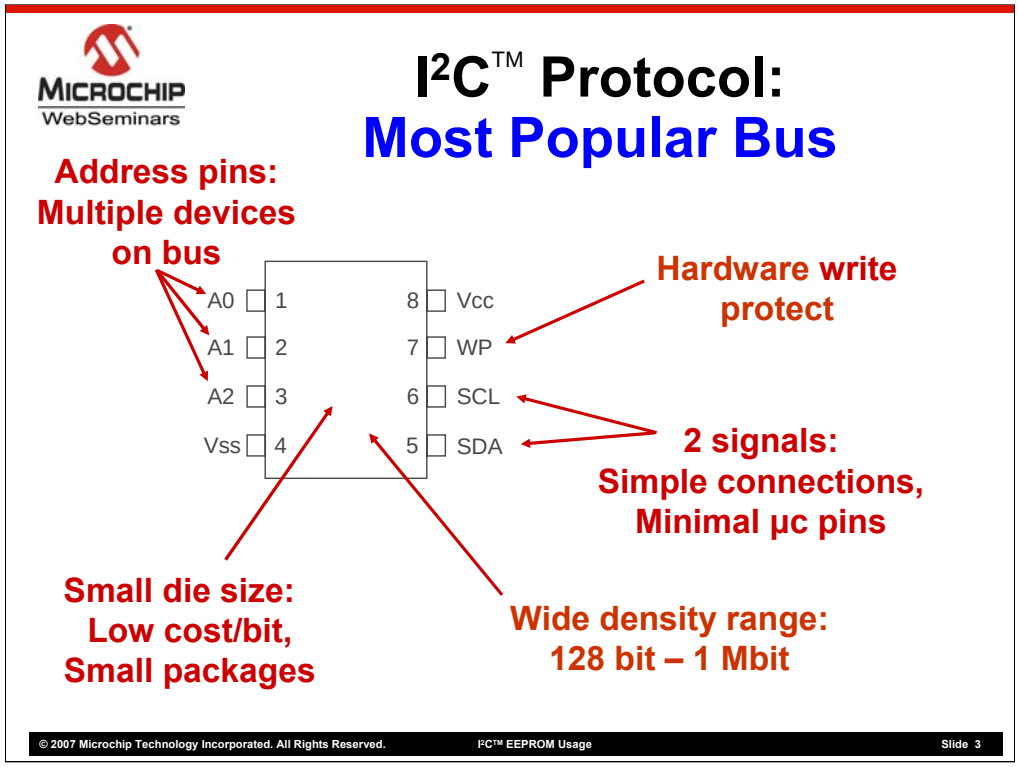
We'll begin this seminar with a summary of the major features of the I²C™ protocol that make it the most popular bus for EEPROMs.

Then we'll look at several hardware recommendations to implement in an I²C system.

Following that, we will focus on several firmware enhancements to boost performance.

We will finish up with a quick summary and some ideas for more places to find information.

In the end, we will have covered 7 specific recommendations. We picked these items since they represent some of the questions most often asked by our customers.



As I mentioned earlier, the I²C™ bus is the most popular of the 3 serial EEPROM protocols; (Microwire and SPI are the other two.) On this slide, I will run through some of the advantages of the bus. Here you can see the typical pinout of an I²C device showing pins 1 through 3 as address pins A0, A1 and A2. Pin 4 is Vss, or ground. Pin 5 is SDA, the data line. Pin 6 is SCL, the clock signal. Pin 7 is write protect, and pin 8 is voltage, or Vcc.

The first advantage of the I²C protocol is that it only requires 2 signal lines: clock and data. The other 2 protocols both require 4 signal lines between the EEPROM and the master device. This means simpler wiring and fewer microcontroller pins for I²C devices.

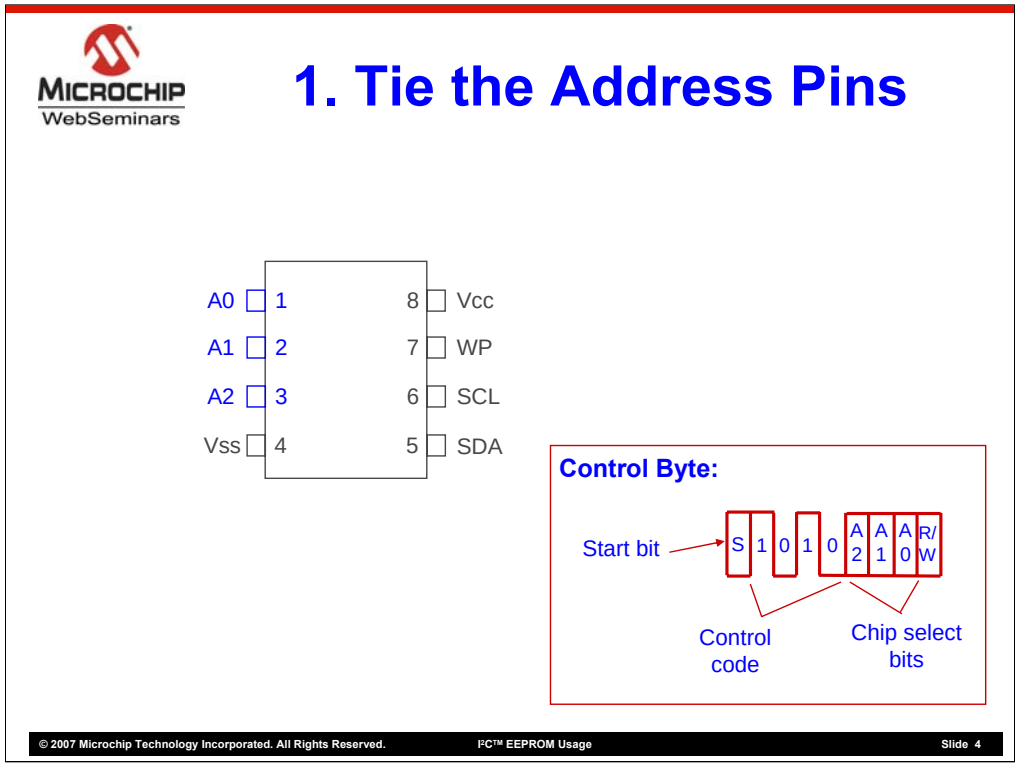
Another advantage is the wide density range. Microchip offers I²C devices from 128 bits all the way up to 1 Mbit – the widest range of any supplier in the industry.

In many cases, the die size for a specific density is smaller for an I²C device than in the other two protocols. That means two benefits: First, a low cost per density. Second, an I²C die can often fit into smaller packages.

Many I²C devices have a write protect pin that prevents writes to the array when activated. We'll talk about an excellent way to use this pin to prevent unwanted writes during power up and power down.

Finally, many I²C EEPROMs include address pins as an easy way to have multiple EEPROMs on a single bus while still only using two connections to the micro.

We will begin our look at hardware recommendations on the next slide.



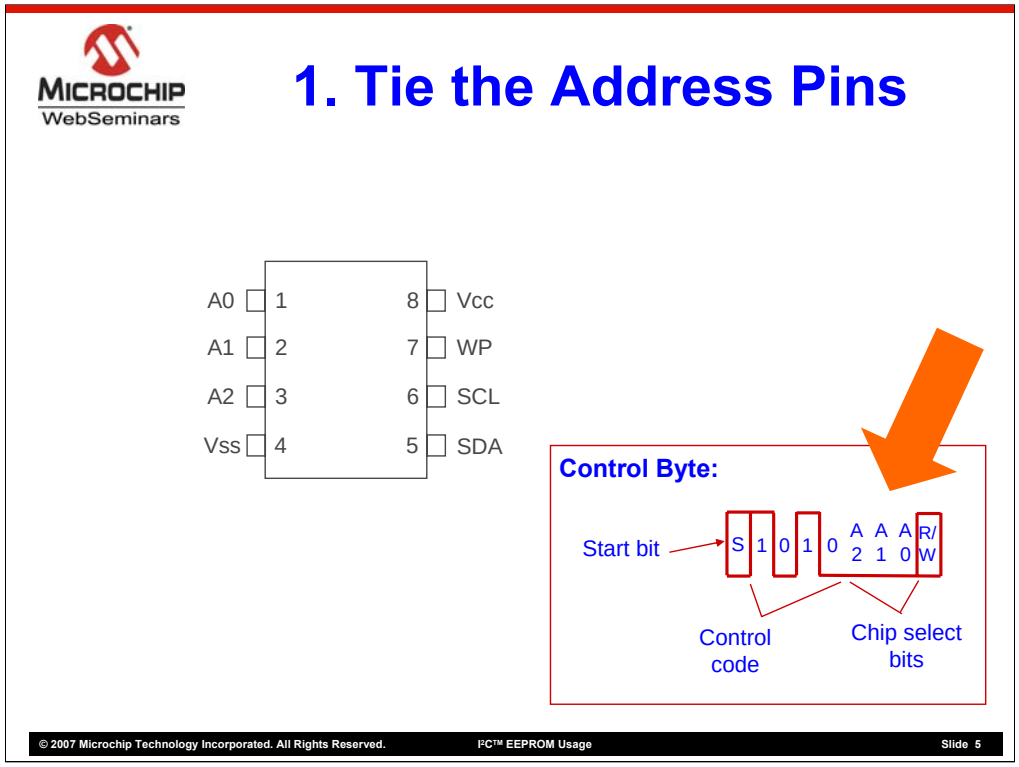
I'm starting this hardware section with the same pinout diagram of a typical I²C™ EEPROM that we saw on the last slide. Over the next few slides, I'll add several hardware connections until we end up with a complete system.

Our first recommendation is that functional address pins should not be left floating.

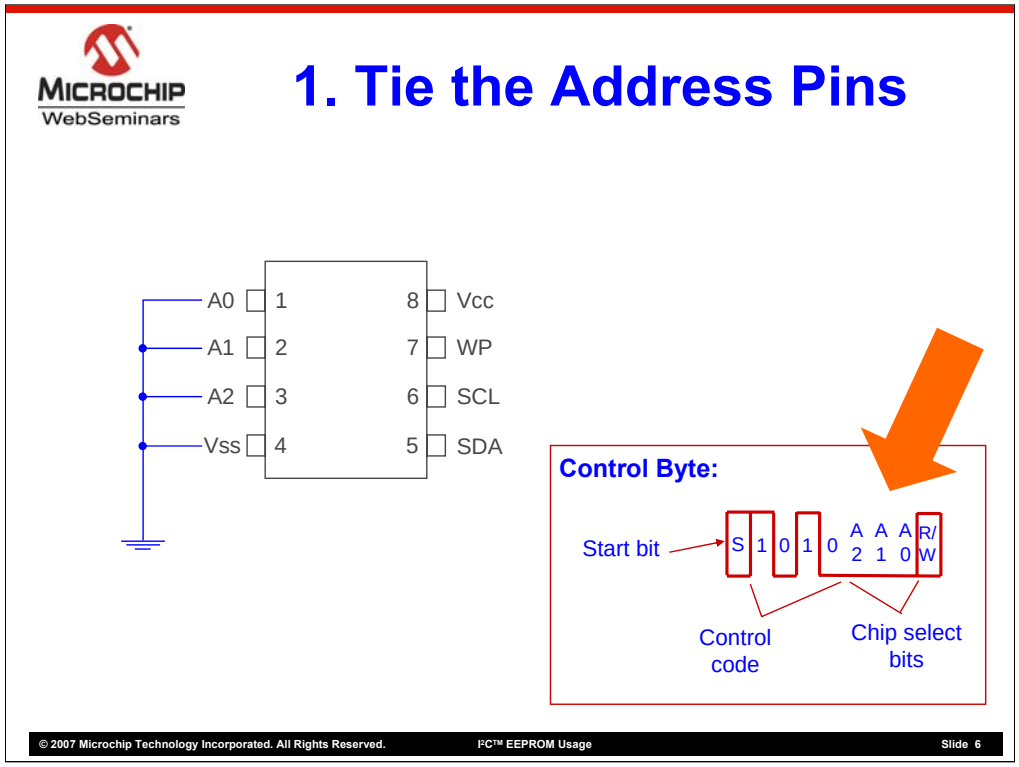
Let's talk a little about how the I²C protocol works and about the function of address pins A0 through A2. The I²C protocol is a 2-wire interface, meaning that the bus has 2 signal lines: clock and data. Every device on the bus connects to both signals. When a master wants to communicate with one of the devices on the bus, it sends a start bit followed by a control byte. You can see a typical EEPROM control byte now. The first 4 bits of the 8-bit control byte are called the control code which is defined by the product data sheet. The master sends the assigned control code to address a certain device on its bus. Generally, serial EEPROMs are assigned a control code of 1010, as shown on the slide. But, how can the master call a specific EEPROM if there is more than one on the bus? That's where the address pins are used.

The chip's address pins and the corresponding chip select bits in the control byte solve this issue by offering a way to have up to 8 EEPROM devices on the bus. Three address pins on the EEPROM can be tied to either Vcc or ground in a total of 8 different combinations. The control byte contains 3 chip select bits that must be programmed to correspond to the logic levels of the A0, A1, and A2 address pins so that the master can specify which EEPROM it wants to address.

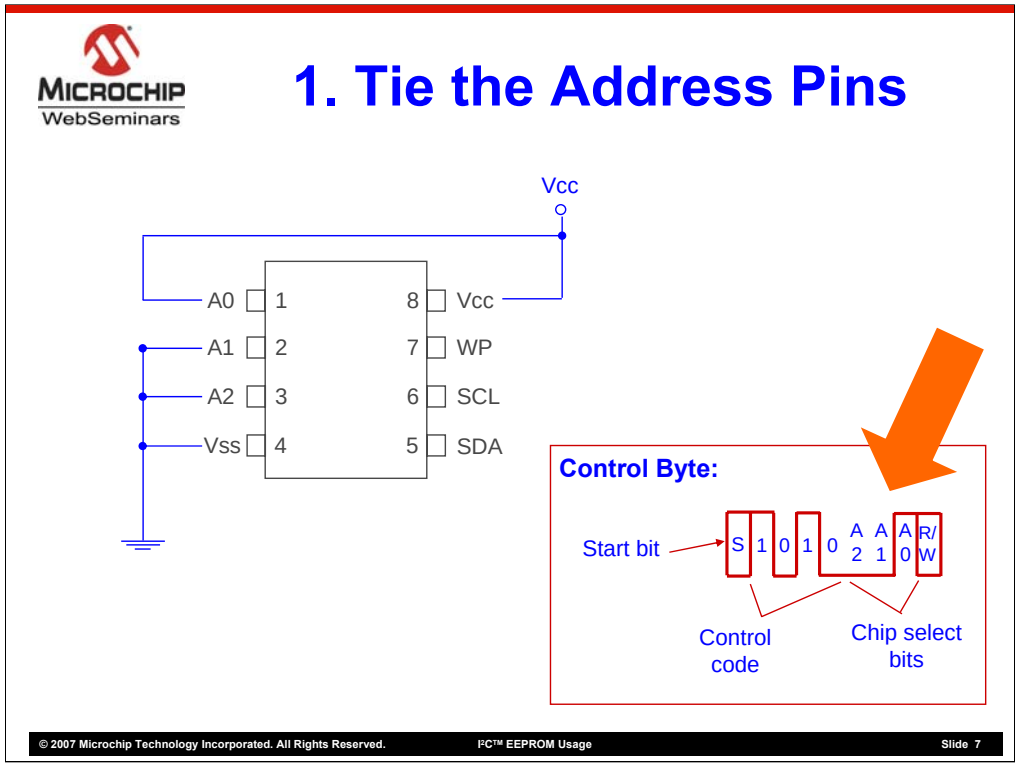
Let's look at an example.



I have just changed my control byte to specify the three chip select bits to all be low, or 000. Since the address pins must correspond to these chip select bits, all the address pins must be tied low so the part will function properly. I'm showing those connections now.



I have just changed my control byte to specify the three chip select bits to all be low, or 000. Since the address pins must correspond to these chip select bits, all the address pins must be tied low so the part will function properly. I'm showing those connections now.

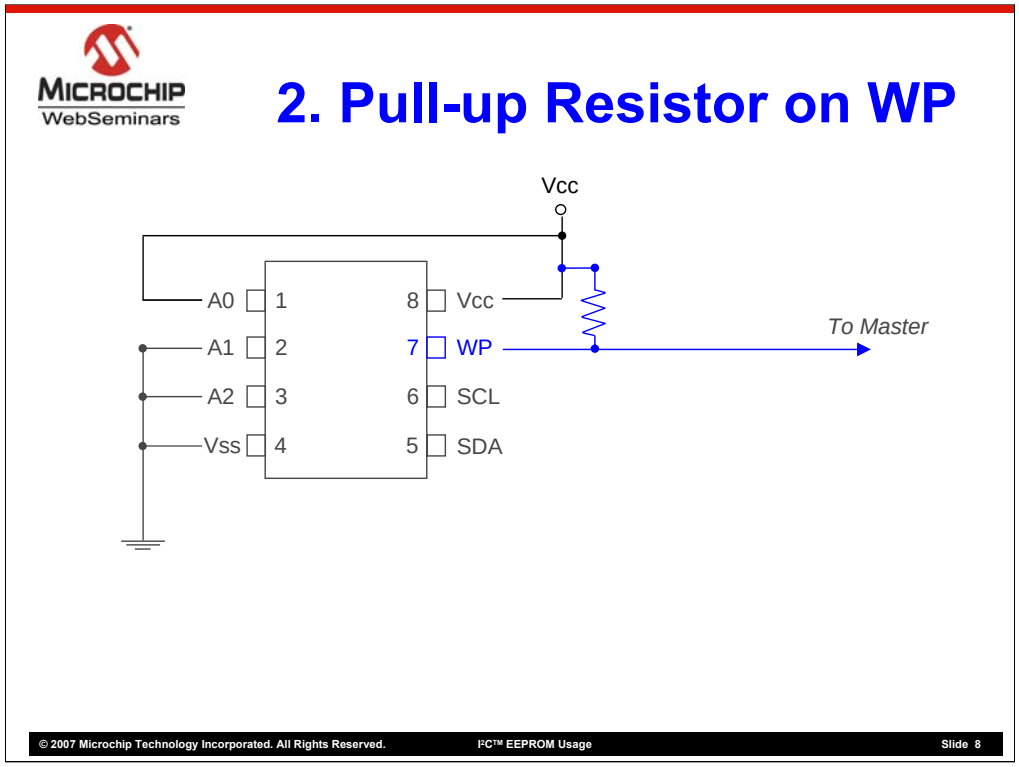


A 2nd EEPROM can be added to the bus, but it must have a different chip-select/address pin combination. To show this, I have reconnected A0 from Vss to Vcc. Note that the chip select bits have also changed to 001 to match the address pins.

Not all devices have 3 functional address pins; some devices have pins that are not internally connected. It is not necessary to connect non-functional address pins to ground or voltage. Check the data sheet to see how many functional address pins a particular device has.

Also, some applications use a microcontroller to drive address pins high or low. Be sure that the inputs are driven to a 1 or 0 before commencing operations.

There are a couple takeaways here. First, check the data sheet to determine the specific address-pin functionality of your device. Next, ensure that the chip select bits match the levels of the address pins. Finally, even if the address pins are not being used, functional address pins must not be left floating.



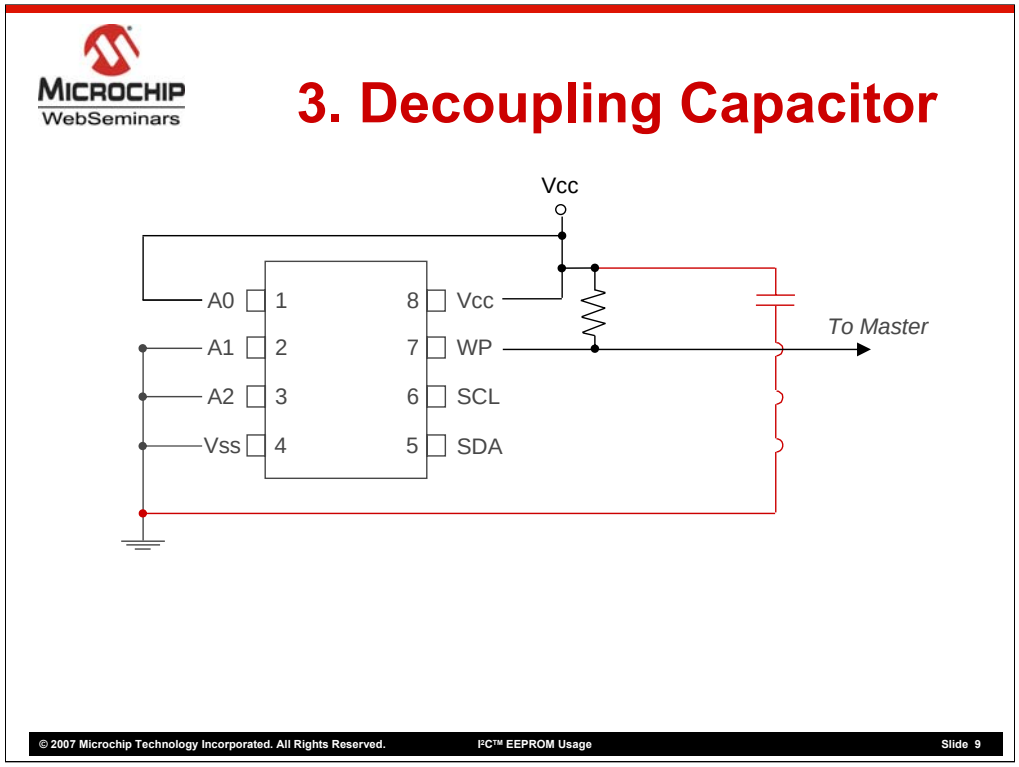
Our 2nd recommendation is to use a pull-up resistor on the write protect pin. One of the advantages of using the I²C™ protocol is this hardware write protect feature. Most I²C EEPROMs have a WP, or write protect, pin here shown as pin 7. This pin can be used to prevent writes to all or a portion of the array, depending on the part. If the pin is at logic 1, write protect is enabled, and writes are not permitted. If WP is at logic 0, writes are permitted.

Many customers simply tie the Write Protect pin to ground and always allow writes to the array. This implementation, however, can be risky since EEPROMs typically write at low voltages, often below 1.8 volts. At the same time, many microcontrollers operate erratically at these low voltages and can send random codes to the EEPROM that could be interpreted as actual commands. If a micro sends an unintended write command, the EEPROM will execute that command, and data may be corrupted. The potential for this occurrence is especially high during power up and power down operations.

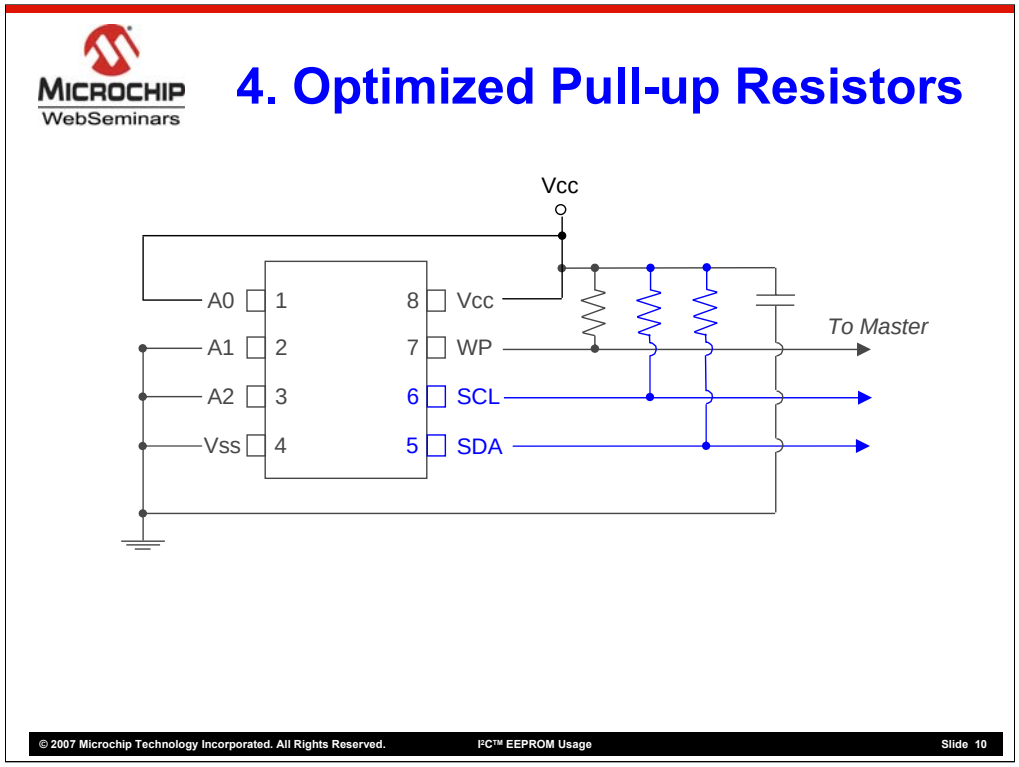
But, there is an easy fix to prevent this. We recommend connecting the write protect pin to the controller along with a pull-up resistor, as I'm showing on the slide now. This set-up prevents writes during power-ups and power-downs by keeping the write protect pin high during these times when the pin is not being explicitly driven.

When a write is actually required, the write protect pin must be driven to logic 0 by the master. This adds a degree of protection to the design by using just one resistor and one microcontroller pin.

If the Write Protect pin is not controlled, it still can not be left floating; it must be tied low which enables writes at all times. Refer to the specific product data sheet for more details on these write protect features.



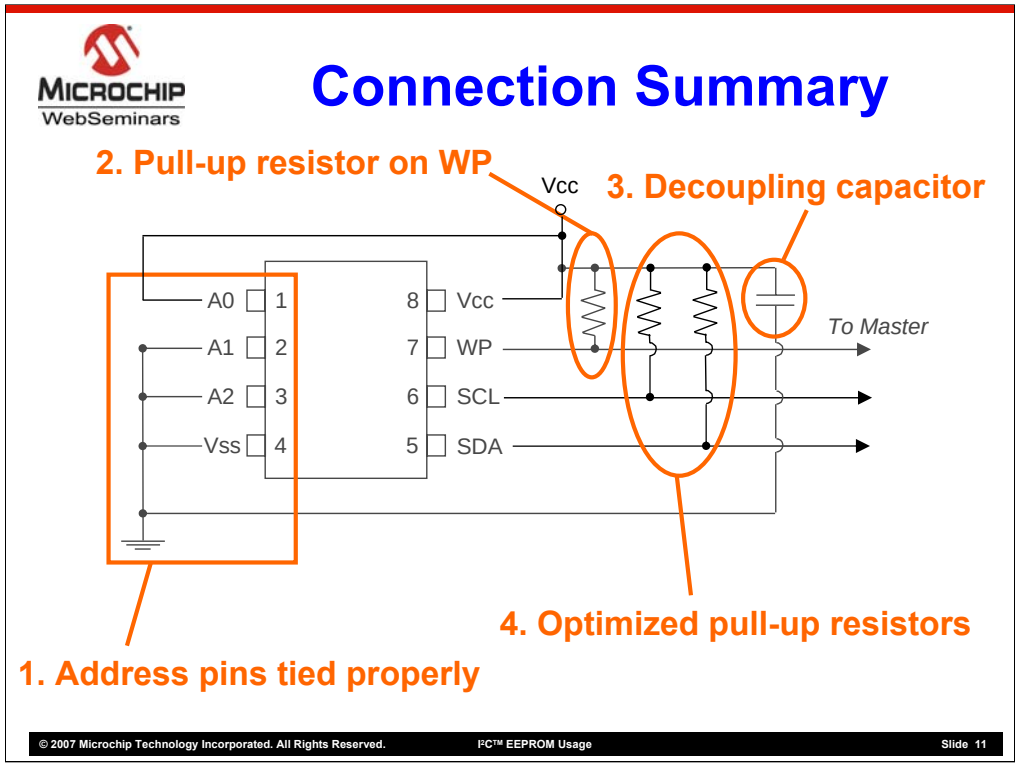
Our 3rd recommendation is a quick one. It is important to remember good engineering practice and add a decoupling capacitor of around 0.1 uF as close to the device as possible to help filter high-frequency noise from the power supply.



This 4th topic is an often overlooked opportunity to optimize an I²C™ EEPROM design. The I²C protocol requires pull-up resistors on both the clock and data lines; I'm adding them to the SCL and SDA pins now.

Here is where the opportunity to improve comes in: Our datasheet suggests either a 10 or 2 kOhm resistor, depending on the speed on the bus. But, these suggestions are for typical applications. If you have a system with high capacitance or where minimizing power consumption is critical, you can change these resistor values to fine tune your system. Our App note AN 1028 goes through the calculations to select the optimum resistor size by taking into account supply voltage, bus capacitance, and input current.

If you want to minimize the power consumption or maximize the speed of a design, you can go through these calculations. I highly recommend looking at our app note if these are concerns of yours.



Now that this slide is finalized, you can see the recommended connections for an I²C EEPROM device:

•First, the address pins are all tied to either ground or Vcc, and the control byte corresponds to the address pin settings

•Second, there is a pull-up resistor on Write Protect to protect the array during indeterminate power levels

•Third, a decoupling capacitor has been added to smooth out power inputs

•Finally, both clock and data lines have pull-up resistors that have been optimized for either speed or current

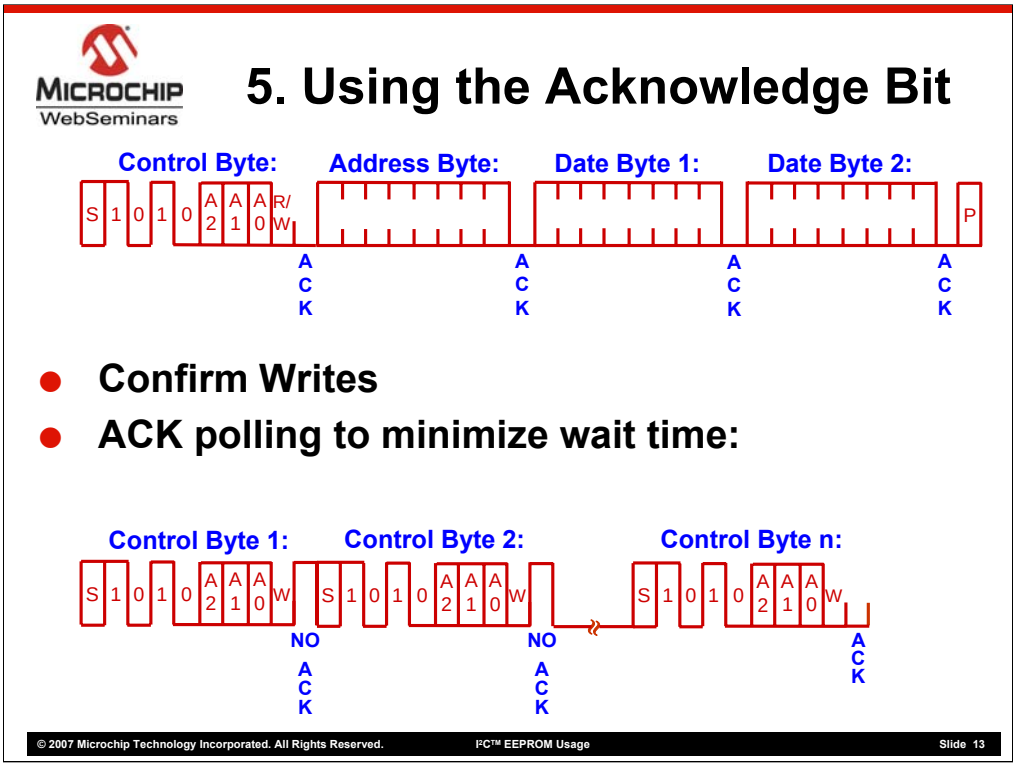
With that, let's go back to our agenda slide.

MICROCHIP WebSeminars

Agenda

- I²C™ Benefits
- Hardware Recommendations
- **Firmware Performance Enhancements**
- For More Information

We just completed the recommended hardware connections portion of this seminar. Now we will look at ways that we can use firmware implementation choices to optimize total system performance.



Our 5th recommendation begins our firmware discussion. We'll now talk about using the acknowledge bit.

I'll be showing an example of a page write command on this slide. You can see the start bit, labeled 's', followed by the control byte. (Note the 1010 control code and the 3 chip select bits, A0 through A2). The last bit of the control byte is the read/write bit, which is logic 0 for a write command. After the EEPROM receives the control byte, it sends a logic 0 signal back to the master. This is called the acknowledge, or ACK, bit.

After the master receives the acknowledge bit, it sends one or more address bytes to the EEPROM. Then the EEPROM sends another ACK bit back to the micro. Each data byte sent by the master is also followed by an ACK bit. The key for this discussion is that after every byte is sent to the EEPROM, the EEPROM sends a logic 0 acknowledge bit back to the master.

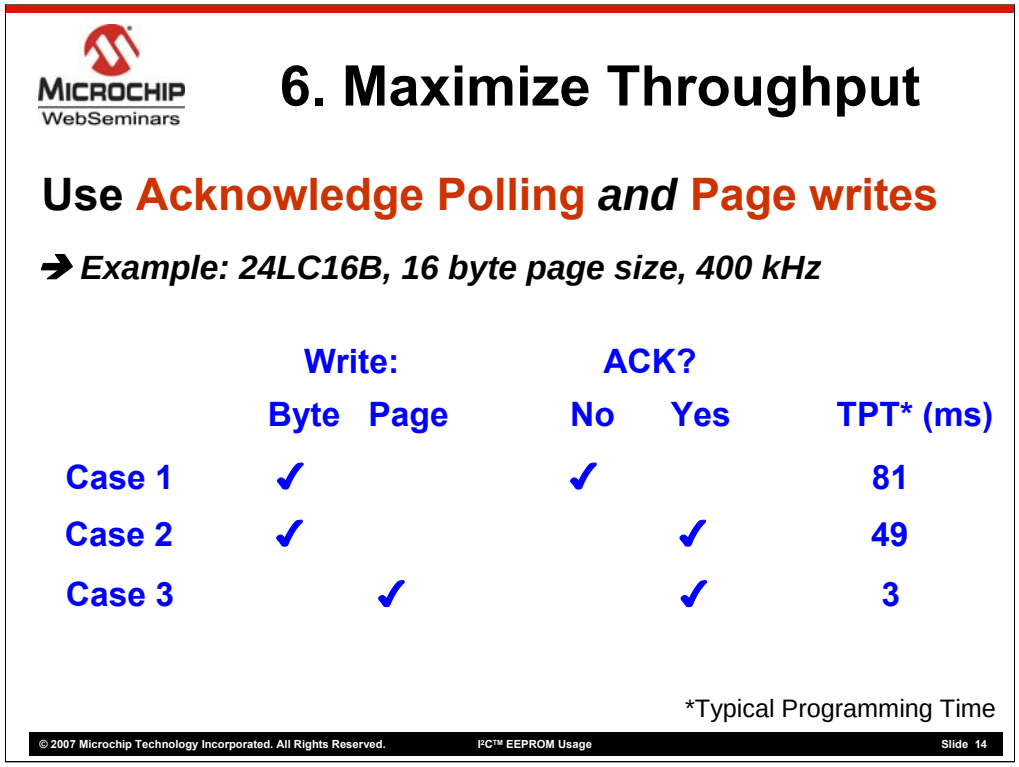
You can use this feature of the I²C specification in 2 ways. The first usage is to quickly find errors by confirming that the EEPROM has actually written data. If the micro does not receive the logic 0 it expects as an acknowledge, it knows that the EEPROM is not responding. App note AN 1028 describes how to implement a software bus reset sequence to deal with these errors.

A second key way to use the acknowledge bit is to implement acknowledge polling, which is a way to reduce the wait time during a write cycle. Let's talk about how an EEPROM completes a write cycle. In the example page write sequence shown here, the EEPROM temporarily stores incoming data bytes in its page buffer. When a stop condition is received, shown by the last bit labeled "P", the data in the buffer is written to the EEPROM's non-volatile array. The time required to complete this transfer is called the write cycle time.

Since the EEPROM will not accept commands during a write cycle, the master must wait until the write cycle is complete before sending the next command. There are basically 2 ways to do this. The first method is to force the controller to wait for the maximum specified write cycle time of the memory device – usually 5-10 ms. But most write cycles take less than that. What if you don't want to wait?

That's when you can use an acknowledge polling routine to find out when the write cycle is complete. To perform acknowledge polling, the master sends a start bit and a control byte to the EEPROM, shown on the bottom of the slide as control byte 1. If the EEPROM is still completing a write cycle, no ACK will be returned. Then the master can resend another start bit/control byte combination, shown as control byte 2. This continues until the write cycle is complete, when the EEPROM will send the ACK bit, as shown in control byte "n." At that point, the master knows that the write cycle is over and can proceed with the next command.

This is an extremely useful, yet underutilized, feature of I²C designs. In the next slide, we'll look at some specifics as to how much time acknowledge polling can actually save.



As we just discussed on the previous slide, acknowledge polling is a simple way to improve data throughput. A second way to improve throughput is to use page writes. Let's go through some numbers to show the improvements that you can get by using these two features.

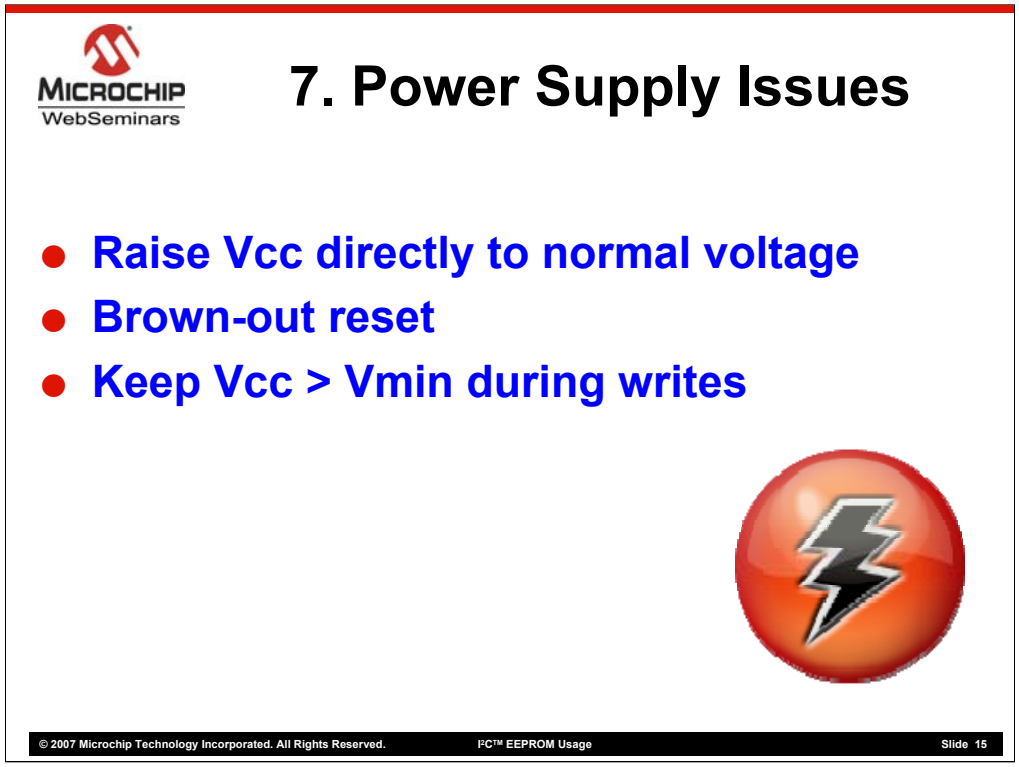
Here's an example. We'll look at a 16 Kbit EEPROM with a page size of 16 bytes at a bus speed of 400 kHz. We will also assume that we need to write a full page of data (16 bytes) to the EEPROM.

The worst-case scenario would be using byte writes, that is writing one byte at a time, with no acknowledge polling. The system must complete 16 write cycles, each lasting the maximum 5 ms write cycle time since there is no acknowledge polling. The typical throughput time is therefore 81 ms to complete the 16 writes.

Now let's add acknowledge polling so that the system does not have to wait the entire maximum 5 ms write time. This decreases each write cycle from 5 ms to closer to 3 ms; so the typical throughput time is decreased from 81 to 49 ms.

Finally, we will add in page writes. Page writes cut down throughput time by writing all 16 bytes at once instead of performing multiple byte writes. The typical write cycle time for this single page write operation is only 3 ms.

As you can see, to completely optimize the system, the designer should use both page writes and acknowledge polling.



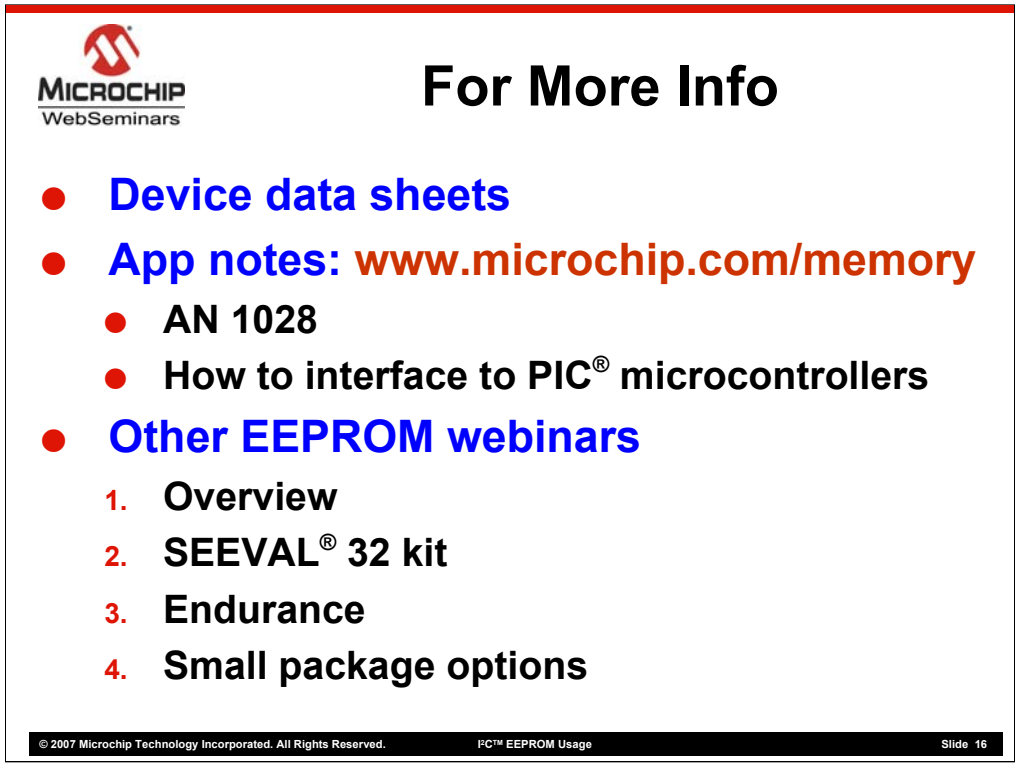
Now we will talk a little more about ways to protect an EEPROM from data corruption during times when the power is outside of the normal operating conditions. Our seventh and final topic is ways to deal with these power issues.

First, on power-up, Vcc should begin at zero and rise directly to its normal voltage. In order to ensure proper power on reset, it is best not to linger at an ambiguous level below the minimum operating voltage.

Next, it is important to understand the brown-out reset operation of both the microcontroller and the EEPROM. There can be cases when the micro has tripped a brown-out threshold, but the EEPROM has not. The best practice in these cases is to make sure the microcontroller and memory devices are both properly reset.

Finally, make sure that the operating voltage is kept above the minimum allowed voltage for the entire write cycle to ensure data integrity.

Once again, these suggestions are detailed in app note AN1028.



We have a lot more information about I²C™ EEPROMs available on our web site.

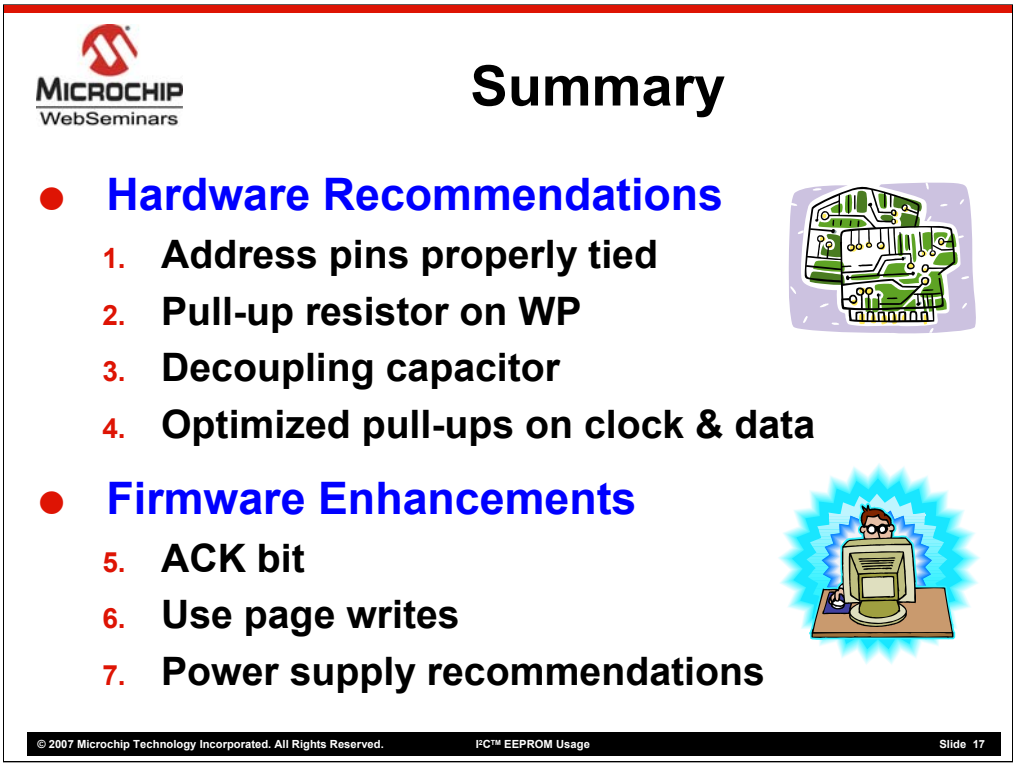
Our data sheets are an excellent source that describe, in great detail, how the I²C spec works.

We also have several educational app notes. I have been referencing AN 1028 throughout this seminar. Between it and the product data sheets, you have an excellent baseline to understand EEPROM operations and recommended design practices.

We also have dozens of app notes explaining how to interface a Microchip I²C EEPROM to many of Microchip's PIC® microcontrollers including PIC10, PIC12, PIC16 and PIC 18 devices. In most cases, these app notes include downloadable source code.

Both the data sheets and app notes can be found at www.microchip.com/memory.

Finally, we have several other web seminars on EEPROMS, including an introductory overview, our SEEVAL® 32 design kit, EEPROM endurance, and the many small package options that are available.



And that completes this web seminar in which we've discussed 7 major recommendations for I²C™ serial EEPROM designs. Let's quickly review those topics now.

Our hardware recommendations are:

First, make sure the address pins are properly tied and are not floating

Next, use a pull-up resistor on the write protect pin to help prevent unwanted writes during power up and power down

3rd, use a decoupling capacitor

4th, consider using optimized pull-ups on both clock and data

The next 3 recommendations are ways to improve performance

Number 5, use the acknowledge bit to look for errors and to minimize throughput time

6, use page writes whenever possible

Lastly, don't forget the power supply and reset recommendations we just went through

To get more details on any of these concepts, please look through our app notes and data sheets.

Thanks a lot for your time.